Cache-Based Architectures for High Performance Computing

D. M. Pressel
Computational and Information Sciences Directorate
U.S. Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5066

Keywords
Cache, supercomputer, high performance computing

Abstract

Many researchers have noted that scientific codes perform poorly on computer architectures involving a memory hierarchy (cache). Furthermore, a number of people and some vendors concluded that simply making the caches larger would not solve this problem. Instead of using large caches, some vendors of high performance computing systems have opted to equip their systems with fast memory interfaces, but with a limited amount of on-chip cache and no off-chip cache (e.g., the Cray T3D, Cray T3E, and the IBM SP with the POWER 2 Super Chip).

Some **RISC**-based high performance systems supported some sort of prefetching or streaming facility that allows one to more efficiently stream data between main memory and the processor (e.g., the Intel Paragon, Cray T3E, IBM SP POWER 2 and P2SC, SGI Origin 2000, etc.). However, there are fundamental limitations on the benefits of these approaches, which makes it difficult to see how these approaches by themselves will solve all of the problems associated with the "Memory Wall." In fact, it has been shown that if one relies solely on this approach for the Cray T3E, one is unlikely to achieve much better than 4–6% of the machine's peak performance.

Does this mean that as the speed of RISC/ **CISC** processors increases, systems designed to process scientific data are doomed to hit the Memory Wall? The answer to that question depends on the ability of programmers to find innovative ways to take advantage of caches. This paper discusses some of the techniques that can be used to overcome this hurdle. Once these techniques have been identified, one can then consider what types of hardware resources are required to support these techniques.[*][†]

[*] This work was made possible by the grant of computer time by the **DOD HPCM** Program. Additionally, it is largely based on work that was funded as part of the **CHSSI** administered by the DOD HPCM Program.
[†] All items in bold type are defined in the glossary.

1. Introduction

Many researchers have noted that scientific codes perform poorly on computer architectures involving a memory hierarchy (cache) (Bailey 1993). Furthermore, as a result of simulation studies, running microbenchmarks on real machines, and running real codes on real machines, a number of people and some vendors concluded that simply making the caches larger would not solve this problem. As a result of these conclusions, some vendors of high performance computing systems have opted to equip their systems with fast memory interfaces, but with a limited amount of on-chip cache and no off-chip cache (e.g., the Cray T3D, Cray T3E, and the IBM SP with the POWER 2 Super Chip).

Some RISC-based high performance systems supported some sort of prefetching or streaming facility that allows one to more efficiently stream data between main memory and the processor (e.g., the Intel Paragon, Cray T3E, IBM SP POWER 2 and P2SC, SGI Origin 2000, etc.). However, there are fundamental limitations on the benefits of these approaches which make it difficult to see how these approaches by themselves will solve all of the problems associated with the “Memory Wall.” In fact, it has been shown that if one relies solely on this approach for the Cray T3E, one is unlikely to achieve much better than 4–6% of the machine’s peak performance (O’Neal and Urbanic 1997).

Does this mean that as the speed of RISC/CISC processors increases, systems designed to process scientific data are doomed to hit the Memory Wall? The answer to that question depends on the ability of programmers to find innovative ways to take advantage of caches. This paper discusses some of the techniques that can be used to overcome this hurdle. Once these techniques have been identified, one can then consider what types of hardware resources are required to support them.

It is important to note that this work is based on two key concepts:

(1) It is acceptable to make significant modifications to the programs at the implementation level.

(2) Not all computer architectures are created equal. Therefore, one will frequently have to define a minimum set of resources that the tuning will be aimed at (e.g., cache size).

2. Caches and High Performance Computing

Many researchers have noted that scientific codes perform poorly on computer architectures involving a memory hierarchy (cache) (Bailey 1993). Furthermore, as a result of simulation studies (Kessler 1991), running microbenchmarks on real machines (Mucci and London 1998), and running real codes on real machines, a number of people and some vendors concluded that simply making the caches larger would not solve this problem. In fact, one group of researchers observed the following:

"For all the benchmarks except cgm, there was very little temporal reuse, and the cache size that had approximately the same miss ratio as streams is proportional to the data set size" (Palacharla and Kessler).

As a result of these conclusions, some vendors of high performance computing systems have opted to equip their systems with fast memory interfaces but with a limited amount of on-chip cache and no off-chip cache. Examples of such conclusions are as follows:

(1) Intel Paragon: 16-kB instruction cache, 16-kB data cache.

(2) Cray T3D: 16-kB instruction cache, 16-kB data cache.

(3) Cray T3E: 8-kB primary instruction cache, 8-kB primary data cache, 96-kB combined instruction/data secondary cache.

(4) IBM SP with the Power 2 Super Chip: 64-kB instruction cache, 128-kB data cache.

Can a way be found to beat these conclusions? If so, how and why are these techniques not used more frequently? The following is a list of techniques that have been used to improve the cache miss rate for a variety of scientific codes:

(1) Reordering the indices of matrices to improve spatial locality.

(2) Combining matrices to improve spatial locality.

(3) Blocking the code to improve both spatial and temporal locality.

(4) Tiling the matrices to improve spatial locality.

(5) Reordering the operations in a manner that will improve the temporal locality of the code.

(6) Recognizing that if one is no longer dealing with a vector processor, it may be possible to eliminate some scratch arrays entirely, while substantially reducing the size of other arrays. When done well, this can increase both the spatial and temporal locality by an order of magnitude.

(7) Writing the code as an out-of-core solver. In many cases, it would not actually be necessary to perform input/output (I/O). However, by restricting the size of the working arrays, in theory, one could significantly decrease the rate of cache misses that miss all the way back to main memory. This method is especially good at improving the temporal locality.

(8) Borrowing the concept of *domain decomposition*, which is frequently used as an approach to parallelizing programs. While this approach is not without its consequences, it can significantly decrease the size of the working set (or help to create one where it would otherwise not exist). Again, this method is aimed at improving the temporal locality.

This demonstrates that there are methods for significantly decreasing the cache miss rate. However, as will be seen later in this paper, some of these techniques work best when dealing with *large* caches. Unfortunately, many of the more popular **MPPs** either lacked caches entirely (e.g., the NCUBE2 and the CM5 when equipped with vector units) or were equipped with small to modest sized caches (e.g., the Intel Paragon, Cray T3D and T3E, and the IBM SP with the POWER2 Super Chip processors). As a result, for many programmers working on high performance computers, there was no opportunity to experiment with ways to tune code for large caches. Furthermore, since many codes are required to be portable across platforms, there was little incentive to tune for architectural features that were not uniformly available.

3. Understanding the Limitations of a Stride-1 Access Pattern

Before continuing, we will briefly discuss spatial and temporal locality. Let us consider the case of an R12000-based SGI Origin 2000 with prefetching turned off a 300-MHz processor generating one load per cycle with a Stride 1 access pattern and no temporal locality (this is an example of pure spatial locality of reference), with a cache line size of 128 bytes for 64-bit data. This arrangement will have a 6.25% cache miss rate. Assuming no other methods of latency hiding are used and assuming a memory latency of 945 ns (Laudon and Lenoski 1997), then this processor will spend 95% of its time stalled on cache misses. Phrasing this another way, if one assumes that the peak speed of the processor is one multiply-add instruction per cycle, the best that the processor will deliver is 32 **MFLOPS** out of a peak of 600 MFLOPS. This result compares favorably with the measured performance in Table 1.

Table 1. Single Processor Results From the Streams Benchmark for Commonly Used HPC Systems[a]

System	Peak Speed (MFLOPS)	TRIAD (MFLOPS)
Cray T3E-900 (Alpha 21164)	900	47.3
Cray T3D (Alpha 21064)	150	14.7
IBM SP P2SC (120 MHz)	480	65.6
IBM SP Power 3 SMP High (222 MHz)	888	51.2
SGI Origin 2000 (R12K - 300 MHz)	600	32.3
SUN HPC 10000 (Ultra SPARC II - 400 MHz)	800	24.7

[a] (McCalpin 2000)

From this, it can seen that for large problem sizes, relying on spatial locality alone will not produce an acceptable level of performance. Instead, one must combine spatial locality with temporal locality (data reuse at the cache level). However, if a vector optimized code is run on this machine with the same assumptions, one can, at best, work on 131,072 values per megabyte of cache (the R12000 based SGI Origin 2000 is currently being sold with 8-MB secondary caches). Table 2 demonstrates where some of the strengths and weaknesses of this approach lie. Clearly the two most important concepts are:

(1) Maximize the processing of the data a grid point at a time.

(2) Minimize the amount of data that needs to be stored in cache at one time (minimize the size of the working set).

Assuming that the techniques mentioned in the previous section have improved the cache miss rate to 1%, then the peak delivered level of performance rises to 157 MFLOPS (or spending 74.9% of the time). Similar results are obtained when analyzing all CISC- and RISC-based architectures. However, only those architectures with large caches lend themselves to some of these tuning techniques.

4. Results

When Karen Heavey of **ARL**, Aberdeen Proving Ground, MD, first attempted to run a 3-million grid point test case with F3D on an SGI Power Challenge (75-MHz R8000 processor - 300 MFLOPS), a 10 time step run took over 5 hours to complete. The same run when run on a Cray C90 takes roughly 10 minutes to complete. There was never a chance of running it that fast on the Power Challenge, since the processor is slower. However, it was hoped that run times of roughly 30 minutes might be achievable. Table 3 lists the adjusted speed of the RISC optimized code when run on a variety of platforms. The speed has been adjusted to remove the startup and termination costs, which are heavily influenced by factors that are not relevant to this discussion.

Table 4 lists the adjusted speed of the RISC optimized code for a variety of problem sizes when run on the SGI Origin 2000 (R12000) and the SUN HPC 10000. Finally, Table 5 lists the dimensions of the grids used for each of these test cases.

5. Prefetching and Stream Buffers vs. Large Caches

Now that it has been established that large caches can be of value, let us consider the relative performance of systems that stressed prefetching and/or a fast low latency memory system versus those that include a large cache. Tables 6 and 7 contain some real world examples of codes that were able to benefit from the presence of a large cache. This is not to say that all codes will benefit form the presence of a large cache. In particular, it is no accident that the version of F3D that was parallelized using compiler directives was able to take advantage of a large cache. It was extensively tuned for such an architecture. Other codes might perform better on the Cray T3E, especially if they were never tuned for cache-based systems.

Table 2. The Size of the Working Set for a 1 Million Grid Point Problem

Problem Description	Number of Variables (Per Grid Point)	Size of the Working Set (Bytes)
1-D	1	8,000
	4	32,000
	30	240,000
	100	800,000
2-D 1000 × 1000 (Processed as a 1-D problem)	1	8,000
	4	32,000
	30	240,000
	100	800,000
2-D 1000 × 1000 (Processed as 2-D vector optimized problem, 1 row or column at a time)	1	8
	4	32
	30	240
	100	800
2-D 1000 × 1000 (Processed one grid point at a time for maximum temporal reuse)	1	8
	4	32
	30	240
	100	800
3-D 100 × 100 × 100 (Processed as a 1-D problem)	1	8,000
	4	32,000
	30	240,000
	100	800,000
3-D 100 × 100 × 100 (Processed as a plane of data at a time as a 1-D problem)	1	80
	4	320
	30	2,400
	100	8,000
3-D 100 × 100 × 100 (Processed as a 3-D vector optimized problem, 1 row or column at a time)	1	800
	4	3,200
	30	24,000
	100	80,000
3-D 100 × 100 × 100 (Processed one grid point at a time for maximum temporal reuse)	1	8
	4	32
	30	240
	100	800
Block of data 32 × 32 (Processed as a block)	1	8,192
	4	32,768
	30	245,760
	100	819,200

Table 3. The Performance of the RISC Optimized Version of F3D for Single Processor Runs for the 3-Million Grid Point Test Case

System Name	Processor	Clock Rate (MHz)	Peak Speed (MFLOPS)	Adjusted Speed (Time Steps/Hour)
Cray C90[a]	Proprietary	238	952	81.
SGI Power Challenge	R8000	75	300	23.
SGI Power Challenge	R10000	195	390	32.
SGI Challenge	R4400	200	100	10.
Convex Exemplar SPP-1600	HP PA 7200	120	240	16.
SGI Origin 2000	R10000	195	390	41.
SGI Origin 2000	R12000	300	600	61.
SUN HPC 10000	Ultra SPARC II	400	800	46.

[a] Cray C90 ran the vector optimized code.

Table 4. The Performance of the RISC Optimized Version of F3D for Single Processor Runs on the SGI Origin 2000 and the SUN HPC 10000 for a Range of Test Cases

Test Case Size (Millions of Grid Points)	Adjusted Speed (Time Steps/Hour)		Adjusted Speed (Time Steps/Million Grid Points-Hour)	
	SGI	SUN	SGI	SUN
1.00	181.	138.	181.	138.
3.01	61.	46.	184.	138.
12.0	11.7	10.6	140.	127.
35.6	4.0	3.4	142.	121.
59.4	2.3	2.1	137.	125.
124.	1.05	0.93	130.	115.
206.	0.62	NA	128.	NA

Note: The SGI Origin was equipped with 128, 300-MHz R12000 processors with 8-MB secondary caches and 2 GB of memory per 2-processor node.

Note: The SUN HPC 10000 was equipped with 64, 400-MHz Ultra SPARC II processors with either 4 or 8 MB of secondary caches (one of our systems was upgraded before the series of runs was finished) and 4 GB of memory per 4-processor node.

Note: There was insufficient memory to run the 206-million grid point test case on the SUN HPC 10000.

Note: See Table 5 for a description of the test cases.

Table 5. A Summary of the Test Cases

Test Case Size (Millions of Grid Points)	JMAX			KMAX	LMAX
	Zone 1	Zone 2	Zone 3		
1.00	15	87	89	75	70
3.01	15	87	89	225	70
12.0	15	87	89	450	140
35.6	29	173	175	450	210
59.4	29	173	175	450	350
124.	43	254	266	450	490
206.	71	421	442	450	490

Note: For historical reasons, there were some differences between the 1- and 3-million grid point test cases. All of the remaining test cases were based on the 3-million grid point test case. Only the 1-million grid point test case has been run out to a converged solution. The remaining test cases were only used for scalability testing.

Table 6. Comparative Performance From Running Two Version of F3D Using 8 Processors With the 1 Million Grid Point Test Case

System	Peak Processor Speed (MFLOPS)	Parallelization Method	Performance (Time Steps/Hour)
SGI R12K Origin 2000	390	Compiler Directives	793
SUN HPC 10000	800	Compiler Directives	999
HP V-Class	1760	Compiler Directives	1632
SGI R12K Origin 2000[a]	600	SHMEM	349
Cray T3E-1200[a]	1200	SHMEM	382
IBM SP[a]	640	MPI	199

[a] Results provided courtesy of Marek Behr, formerly of the Army High Performance Computing Research Center (AHPCRC).

Table 7. Comparative Performance From Running the Department of Energy (DOE) Parallel Climate Model (PCM) Using 16 Processors[a,b]

System	Peak Processor Speed (MFLOPS)	Performance (MFLOPS/PE)
SGI R10K Origin 2000	500	60
Cray T3E-900	900	38

[a] This data is based on runs done using the T42L18 test case.
[b] Bettge et al. 1999.

6. Prefetching and Stream Buffers in Combination With Large Caches

Previously, this paper pointed out the limitation of relying solely on prefetching and stream buffers. However, there is also a problem with relying solely on caches, even large caches, to solve all of the performance problems. In particular, there is no reason to believe that as the processor speed increases, the cache miss rate will automatically decrease. Even if one were to increase the sizes of the cache while increasing the speed of the processor, it would seem unlikely that the cache miss rate would significantly decline. (As Table 2 demonstrates, the cache miss rate is a function of the size of the cache and the size of the working set. Once the working set comfortably fits in cache, additional increases in the size of the cache will be of minimal value.) If the memory latency is kept constant, then the gain in performance from increasing the speed of the processor will be sublinear. Table 8 shows an example of this. This is what is known as the *Memory Wall*.

Table 8. The Predicted Performance Increase Resulting From Upgrading a 195-MHz R10000 Processor to a 300-MHz R12000 Processor in an SGI Origin 2000

Percentage of Time Spent on Cache Misses (R10K)	Speedup (Percent)
0	54
10	46
25	36
50	21
75	10
90	4
100	0

However, there is nothing that says one cannot combine both caches and some form of prefetching/stream buffers. The goal of this would not be to prefetch values far enough in advance that they would arrive prior to the time needed. With latencies of over 100 cycles, such a design would effectively be a vector processor such as the Cray SV1. We are also not trying to emulate a vector processor's ability to stream in a large vector of data while encountering the cost of a single cache miss. Instead, the goal is to overlap two or more memory latencies, thereby effectively decreasing the average latency by a factor of two or more.

Before considering some results, let us quickly review the following potential limitations of prefetching:

(1) The hardware and/or compilers may not support it.

(2) The compiler support may be limited (this appears to be the case with the SGI compilers).

(3) With a processor capable of out-of-order execution (e.g., those used by SGI), the hardware may be capable of overlapping two or more cache misses on its own. This makes it difficult to measure an incremental benefit from turning on prefetching.

(4) There may be limitations in the memory system (e.g., bandwidth onto the processor, main memory bandwidth, or interconnect bandwidth) that will limit the potential benefits of turning on prefetching.

(5) No system supports an unlimited number of outstanding prefetches and/or cache misses.

(6) The smaller the cache miss rate, the harder it will be for either the compiler or the hardware to expose opportunities for prefetching. In other words, inefficient codes will benefit the most, but they will still be inefficient.

Table 9 discusses the benefits of using prefetching on the SUN HPC 10000 and on the HP V-Class system with the PA-8500 processor. This is particularly relevant, since currently, the HP system uses one of the fastest microprocessors in production (peak speed is rated at 1760 MFLOPS/processor). As can be seen, there was little or no benefit from using prefetching on the SUN system. A few subroutines ran slightly faster, while others ran slightly slower; but the effect was not judged to be large enough to compile the two groups of subroutines separately. However, on the HP system, the benefit was very significant.

Table 9. The Effect of Turning Prefetching on When Running F3D With the 1-Million Grid Point Test Case on the SUN HPC 10000 and the HP V-Class Systems

System Name	Peak Speed (MFLOPS)	Adjusted Speed No Prefetching	(Time Steps/Hr) Prefetching	Speedup (%)
SUN HPC 10000	800	138	119	🕷14
HP V-Class	1760	150	212	41

7. Conclusions

It is possible to tune some scientific codes to take good advantage of systems with a memory hierarchy. It appears as though two- and three-dimensional problems have an inherent advantage to one-dimensional problems. Also, algorithms that do a lot of work per time step (e.g., implicit **CFD** codes) but exhibit a rapid rate of convergence may be better suited for use with caches than algorithms that do very little work per time step but require a large number of time steps to generate an answer (e.g., explicit CFD codes). In any case, one should be prepared to spend a significant amount of time and effort retuning the code.

On a side note, a surprising outcome of this work is that **BLAS** 1, and, to a lesser extent, BLAS 2, subroutines should be avoided when working with systems that use cache. The BLAS 1 subroutines have little or no ability to optimize for either spatial or temporal locality if it does not already exist. The BLAS 2 subroutines can generate spatial locality through the use of blocking but are inherently unlikely to support temporal locality since they operate on planes of data. Similarly, it was shown that other programming styles that were commonly used with vector processors are distinctly suboptimal for the newer systems. Therefore, while some researchers have expressed a strong desire to maintain a single code for use with both RISC- and vector-based systems, it appears as though this is not a good idea.

To the increasing extent when designing or buying a computer for high performance computing, the correct choice when faced with the choice of *large cache* or *prefetching/stream buffers* will be both. Of course, this assumes that the rest of the system is compatible with that choice.

Acknowledgments

The author would like to thank Marek Behr for giving him permission to use his results in this paper. He would also like to thank the entire CHSSI CFD-6 team for their assistance in his work as part of that team. Finally, he would like to thank his many colleagues that have graciously assisted him in all aspects of the preparation of this paper.

Glossary

AHPCRC -	Army High Performance Computing Research Center
ARL -	U.S. Army Research Laboratory
BLAS -	Basic linear algebra subroutines
CFD -	Computational fluid dynamics
CHSSI -	Common High Performance Computing Software Support Initiative

CISC -	Complicated instruction set computer
DOD -	Department of Defense
DOE -	Department of Energy
HPCM -	High performance computing modernization
MFLOPS -	Million floating point operations per second
MPP -	Massively parallel processor
RISC -	Reduced instruction set computer

References

Bailey, D. H. "RISC Microprocessors and Scientific Computing." *Proceedings for Supercomputing 93*, IEEE Computer Society Press and ACM, 1993.

Bettge, T., A. Craig, R. James, W. G. Strand, Jr., and V. Wayland. "Performance of the Parallel Climate Model on the SGI Origin 2000 and the Cray T3E." *The 41st Cray Users Group Conference*, Minneapolis, MN: Cray Users Group, May 1999.

Kessler, R. E. "Analysis of Multi-Megabyte Secondary CPU Cache Memories." University of Wisconsin, Madison, WI, ftp://ftp.cs.wisc.edu/markhill/Theses/richard-kessler-body.pdf, 1991.

Laudon, J., and D. Lenoski. "The SGI Origin: A ccNUMA Highly Scalable Server." *Proceedings for the 24th Annual International Symposium on Computer Architecture*, New York: ACM, 1997.

McCalpin, J. D. "STREAM Standard Results." http://www.cs.virginia.edu/streams/standard/MFLOPS.html, 2000.

Mucci, P. J., and K. London. "The Cache Bench Report." CEWES MSRC/PET TR/98-25, Vicksburg, MS, 1998.

O'Neal, D., and J. Urbanic. "On Performance and Efficiency: Cray Architectures." Pittsburgh Supercomputing Center, http://www.psc.edu/~oneal/eff/eff.html, August 1997.

Palacharla, S., and R. E. Kessler. "Evaluating Stream Buffers as a Secondary Cache Replacement." *Proceedings for the 21st Annual International Symposium on Computer Architecture*, Los Alamitos, CA: IEEE Computer Society Press.